This listing of claims will replace all prior versions, and listings, of claims in the application.

Listing of Claims:

1-14. (Canceled)

15. (Previously presented) A method for scheduling tasks for processing by a coprocessor, comprising:

gathering tasks for processing by a coprocessor into a user mode command buffer memory group, said tasks relating to a first application, wherein at least a portion of the user mode command buffer is allocated in a context of said first application;

delivering the tasks to a scheduler wherein scheduler functions include determining an order for processing the tasks wherein the order may include tasks that relate to one or more other applications;

determining an order for processing the tasks wherein the order accounts for any relative priority among the first application relating to said tasks and one or more other applications relating to additional tasks, and a corresponding amount of processing time that the first application and one or more other applications are entitled to;

preparing tasks for processing by ensuring that any needed memory resources are available in a coprocessor-accessible memory location wherein the preparing tasks occurs in the order determined by the scheduler; and

submitting tasks prepared according to the preparing to the coprocessor for processing.

16. (Original) A method according to claim 15 wherein the coprocessor includes a graphics processing unit (GPU).

17. (Original) A method according to claim 15, further comprising calling an Application Program Interface (API) when the first application has one or more tasks that require processing by the coprocessor.

18. (Previously presented) A method according to claim 17, further comprising calling a user mode driver wherein the functions of the user mode driver include placing rendering commands associated with the one or more tasks in the user mode command buffer memory group.

19. (Original) A method according to claim 18, further comprising returning the rendering commands to the API, and submitting them to a coprocessor kernel.

20. (Previously Presented) A method according to claim 15, further comprising generating a Direct Memory Access (DMA) buffer by a kernel mode driver wherein one or more tasks that require processing by the coprocessor are used to generate the DMA buffer, and the DMA buffer represents the one or more tasks used to generate the DMA buffer.

21. (Original) A method according to claim 20, further comprising generating a list of memory resources by the kernel mode driver wherein the memory resources represented by the list are needed by the coprocessor to process one or more tasks represented by the DMA buffer.

22. (Original) A method according to claim 21, further comprising building a paging buffer for bringing the memory resources on the list of memory resources to correct memory addresses within the coprocessor-accessible memory location.

23. (Original) A method according to claim 15 wherein said preparing is accomplished by a preparation thread which calls a memory manager process capable of determining a location in the coprocessor-accessible memory location to page any needed memory resources.

24. (Original) A method according to claim 23, further comprising splitting a DMA buffer when the memory manager process determines that there is not enough room in the coprocessor-accessible memory location to page all needed memory resources.

25. (Previously presented) The method of claim 15, wherein computer executable instructions for performing said method are stored on a computer readable medium.

26. (Previously presented) The method of claim 15, wherein computer executable instructions for performing at least a portion of said method are received via a modulated data signal.

27. (Previously presented) The method of claim 15, wherein said method is carried out on a computing device.

28. (Previously presented) A method for scheduling tasks for processing by a coprocessor, comprising:

gathering tasks for processing by a coprocessor into a user mode command buffer memory group, said tasks relating to a first application, wherein at least a portion of the user mode command buffer is allocated in a context of said first application;

delivering the tasks to a scheduler wherein the functions of the scheduler include determining an order for processing the tasks wherein the order may include tasks that relate to one or more other applications;

determining an order for processing the tasks wherein the order accounts for any relative priority among the first application relating to said tasks and one or more other applications relating to additional tasks, and a corresponding amount of processing time that the first application and one or more other applications are entitled to;

preparing tasks for processing by ensuring that any needed memory resources are available in a coprocessor-accessible memory location wherein the preparing tasks occurs in the order determined by the scheduler; and

submitting tasks to the coprocessor for processing;

managing the coprocessor accessible memory to apportion the coprocessor accessible memory among the various tasks; and

providing a per-context virtual address space for the tasks.

29. (Original) A method according to claim 28 wherein the coprocessor is a graphics processing unit (GPU).

30. (Original) A method according to claim 28, further comprising storing a task in a DMA buffer wherein the storing is accomplished by a user mode driver.

31. (Original) A method according to claim 30, further comprising validating a memory resource referenced in a resource list that is associated with the DMA buffer wherein validating entails finding a range of coprocessor-readable memory that is free and asking the kernel mode driver to map a page table or a memory resource handle to that range.

32. (Original) A method according to claim 28 wherein the virtual address space is virtualized through the use of a flat page table that divides coprocessor-readable memory into pages of a predefined memory amount wherein further a page table is provided in the virtual address space that contains identifiers for specifying coprocessor-readable memory addresses.

33. (Original) A method according to claim 28 wherein the virtual address space is virtualized through the use of a multi-level page table that divides coprocessor-readable memory into pages of a predefined memory amount wherein further a multiple page tables are provided in the virtual address space that contain identifiers for specifying coprocessor-readable memory addresses.

34. (Original) A method according to claim 28 wherein a portion of coprocessor readable memory is used to indicate whether all required memory resources associated with a task that requires processing are available in coprocessor-readable memory.

35. (Previously presented) The method of claim 28, wherein computer executable instructions for performing said method are stored on a computer readable medium.

36. (Previously presented) The method of claim 28, wherein computer executable instructions for performing at least a portion of said method are received via a modulated data signal.

37. (Previously presented) The method of claim 28, wherein said method is carried out on a computing device.

38. (Original) A method according to claim 28, further comprising:

assigning a base address for a display surface wherein the display surface is allocated contiguously in coprocessor-readable memory; and

delivering a task to the scheduler wherein processing the task will reassign the base address for a display surface.

39. (Original) A method according to claim 38 wherein processing the task will reassign the base address for a display surface immediately.

40. (Original) A method according to claim 38 wherein processing the task will reassign the base address for a display surface upon the occurrence of a subsequent display synchronization period.

41. (Previously presented) An apparatus for supporting scheduling of tasks for processing by a coprocessor, comprising:

a central processing unit (CPU);

a coprocessor;

one or more applications that generate tasks for processing by the coprocessor, wherein the tasks are first stored in a user mode command buffer, and wherein said tasks are stored in a per-application context in said user mode command buffer; and

a scheduler process for determining an order in which the tasks are processed,

wherein the order accounts for any relative priority among a first application relating to a first set of tasks and one or more other applications relating to additional tasks, and

wherein the order accounts for a corresponding amount of processing time that the first application and the one or more other applications are entitled to.

42. (Original) An apparatus according to claim 41 wherein the coprocessor is a GPU.

43. (Original) An apparatus according to claim 41 wherein the coprocessor supports interruption during the processing of a task by automatically saving task information to a coprocessor-accessible memory location.

44. (Original) An apparatus according to claim 43, further comprising at least one of a private address space for one or more tasks, a private ring buffer where tasks are accumulated, and a private piece of coprocessor-accessible memory where a hardware state is saved when a task is not being processed.

45. (Original) An apparatus according to claim 41 wherein the coprocessor is capable of storing information regarding the history of coprocessor switches from task to task in a specified system memory location readable by the scheduler process.

46. (Original) An apparatus according to claim 45 wherein the coprocessor specifies a base address for the system memory location prior to storing information regarding the history of coprocessor switches from task to task in the system memory location.

47. (Original) An apparatus according to claim 45 wherein the coprocessor specifies a size for the system memory location prior to storing information regarding the history of coprocessor switches from task to task in the system memory location.

48. (Original) An apparatus according to claim 45 wherein the coprocessor specifies a write pointer for indicating where in the system memory location the coprocessor should write to next.

49. (Original) An apparatus according to claim 41 wherein the coprocessor supports fence instructions that cause the coprocessor to write a piece of data associated with a fence instruction at an address specified in the fence instruction.

50. (Original) An apparatus according to claim 41 wherein the coprocessor supports trap instructions that are capable of generating a CPU interrupt when processed by the coprocessor.

51. (Original) An apparatus according to claim 41 wherein the coprocessor supports enable/disable context switching instructions such that when context switching is disabled, the coprocessor will not switch away from a current coprocessor task.

52-65. (Canceled)

66. (Previously presented) A coprocessor for use in connection with a coprocessing scheduler, comprising:

a coprocessor for processing tasks that are initially gathered in a user mode command buffer memory group, wherein said tasks are stored in a per-application context in said user mode command buffer,

wherein said tasks are submitted to the coprocessor by a scheduler process that submits tasks to the coprocessor according to a priority of applications relating to said tasks and that request processing of the tasks, and

wherein the priority determines the amount of coprocessor time one or more applications are entitled to.

67. (Canceled)

68. (Original) A coprocessor according to claim 66 wherein the coprocessor stores information related to a task in a per-context address space, and wherein further the information related to a task allows the coprocessor to process the task or a portion of the task after processing one or more intervening tasks.

69. (Original) A coprocessor according to claim 66 wherein the coprocessor processes tasks from a run list by switching immediately to a subsequent task on the run list when a switching event occurs.

70. (Original) A coprocessor according to claim 69 wherein a switching event comprises at least one of a completion of processing a previously submitted task, a page fault in processing a

task, a general protection fault in processing a task, and a request by a central processing unit (CPU) to switch to a Previously presented run list.

71. (Original) A coprocessor according to claim 66 wherein the coprocessor comprises a GPU.

72. (Original) A coprocessor according to claim 66 wherein the coprocessor accesses memory resources in a coprocessor-readable memory by a memory manager.

73. (Original) A coprocessor according to claim 72 wherein the memory resources comprise references to virtual memory addresses.

74-79. (Canceled)

80. (Previously presented) A method for scheduling tasks for processing by a coprocessor, comprising:

receiving at an Application Programming Interface (API) calls from a first application, said calls requesting tasks requiring processing by a coprocessor;

sending by said API said tasks to a user mode driver for storage in a command buffer memory group, wherein at least a portion of the command buffer is allocated in a context of said first application;

receiving by said API said tasks from said command buffer pursuant to a flush of said command buffer;

sending by said API said tasks to a coprocessor kernel for processing;

delivering the tasks, by the coprocessor kernel, to a coprocessor scheduler wherein coprocessor scheduler functions include determining an order for processing the tasks, wherein the order may include tasks that relate to one or more other applications, wherein the order accounts for any relative priority among the first application relating to said tasks and the one or more other applications relating to additional tasks and a corresponding amount of processing time that the first application and the one or more other applications are entitled to;

preparing tasks for processing by ensuring that any needed memory resources are available in a coprocessor-accessible memory location wherein the preparing tasks occurs in the order determined by the coprocessor scheduler; and

submitting the prepared tasks to the coprocessor for processing.

81. (Previously presented) The method of claim 80 wherein the API operates in conformity with a Direct3D Runtime API.

82. (Previously presented) The method of claim 80 wherein the coprocessor kernel operates in conformity with a DirectX Kernel.

83. (Previously presented) The method of claim 80, further comprising generating a Direct Memory Access (DMA) buffer by a kernel mode driver wherein one or more tasks that require processing by the coprocessor are used to generate the DMA buffer, and the DMA buffer represents the one or more tasks used to generate the DMA buffer.

84. (Previously presented) The method of claim 83, further comprising generating a list of memory resources by the kernel mode driver wherein the memory resources represented by the list are needed by the coprocessor to process one or more tasks represented by the DMA buffer.

85. (Previously presented) The method of claim 84, further comprising building a paging buffer for bringing the memory resources on the list of memory resources to correct memory addresses within the coprocessor-accessible memory location.\

86. (Previously presented) A computer readable storage medium comprising computer executable instructions for scheduling tasks for processing by a coprocessor, comprising:

instructions for receiving at an Application Programming Interface (API) calls from a first application, said calls requesting tasks requiring processing by a coprocessor;

instructions for sending by said API said tasks to a user mode driver for storage in a command buffer memory group, wherein at least a portion of the command buffer is allocated in a context of said first application;

instructions for receiving by said API said tasks from said command buffer pursuant to a flush of said command buffer;

instructions for sending by said API said tasks to a coprocessor kernel for processing;

instructions for delivering the tasks, by the coprocessor kernel, to a coprocessor scheduler wherein coprocessor scheduler functions include determining an order for processing the tasks, wherein the order may include tasks that relate to one or more other applications, wherein the order accounts for any relative priority among the first application relating to said tasks and the one or more other applications relating to additional tasks and a corresponding amount of processing time that the first application and the one or more other applications are entitled to;

instructions for preparing tasks for processing by ensuring that any needed memory resources are available in a coprocessor-accessible memory location wherein the preparing tasks occurs in the order determined by the coprocessor scheduler; and

instructions for submitting the prepared tasks to the coprocessor for processing.

87. (Previously presented) The computer readable storage medium of claim 86 wherein the API operates in conformity with a Direct3D Runtime API.

88. (Previously presented) The computer readable storage medium of claim 86 wherein the coprocessor kernel operates in conformity with a DirectX Kernel.

89. (Previously presented) The computer readable storage medium of claim 86, further comprising instructions for generating a Direct Memory Access (DMA) buffer by a kernel mode driver wherein one or more tasks that require processing by the coprocessor are used to generate the DMA buffer, and the DMA buffer represents the one or more tasks used to generate the DMA buffer.

90. (Previously presented) The computer readable storage medium of claim 89, further comprising instructions for generating a list of memory resources by the kernel mode driver wherein the memory resources represented by the list are needed by the coprocessor to process one or more tasks represented by the DMA buffer.

91. (Previously presented) The computer readable storage medium of claim 90, further comprising instructions for building a paging buffer for bringing the memory resources on the list of memory resources to correct memory addresses within the coprocessor-accessible memory location.